

**SYSTEMS AND METHODS FOR TRANSMITTING
MOTION CONTROL DATA**

5

RELATED APPLICATIONS

This application claims priority of U.S. Provisional Patent
Application Serial No. 60/260,261, which was filed on January 4, 2001.

10

TECHNICAL FIELD

The present invention relates to systems and methods for
transmitting motion control data and, more particularly, to systems and
methods for facilitating the transmission of motion control data from a data
source to a known or unknown client motion control device through a
communications network.

15

BACKGROUND OF THE INVENTION

20

Motion control systems are often used in industrial settings to
perform repetitive, well-defined tasks such as welding, parts assembly,
and the like. Motion control systems have also been used in non-industrial
settings in the form of toys, appliances, and the like for residential use.

25

The specific motion task to be performed by a given motion control
system is defined by motion control data. Motion control data is a set of
instructions conventionally written in a hardware dependent software
language, but systems and methods now exist for creating hardware
independent motion control data. In the following discussion, the term
"application program" will be used to refer to a particular set of motion

30

control data. The terms "application programmer" or "programmer" will be used to refer to the person who writes the application program.

Motion control systems typically employ a motion control device that converts the motion control data into physical movement. Often, the motion control device is connected to a general purpose computer that stores application programs and transfers these programs to the motion control device. In the following discussion, the person responsible for a given motion control device will be referred to as the system operator.

In both industrial and non-industrial settings, application programs are often written by the application programmer at a source location and then run on a motion control system at a remote location. In some situations, the application program is transferred from the source to the destination over a communications network such as the Internet.

From the perspective of the application programmer, the details of the motion control system can be either known or unknown. In addition, the application programmer may or may not know the details of the communications network over which the motion control data is transferred.

One scenario of particular relevance to the present invention is the situation in which an application programmer writes an application program for a given motion task where the programmer does not know or does not want to be limited to the details of a particular motion control system. In particular, the details of the software platform and motion control device(s) may be unknown to the programmer, or the system operator may wish to have the flexibility to change one or both of the software platform and motion control device in the future.

The need thus exists for systems and methods that facilitate the transmission of motion control data from a source to a motion control system over a communications network. The present invention is of particular significance when the details of the motion control system are unknown to the application programmer.

SUMMARY OF THE INVENTION

The present invention may be embodied as a system for transferring a service request between a client application and a motion control system using a communications network. The system comprises a client build module and a service request format module. The client build module builds a service request envelope for containing the service request. The service request is associated with a service performed by the motion control system. In addition, the service request envelope may be transmitted across the communications network. The service request format module extracts the service request from the service request envelope and transmits the service request to the motion control system.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. 1A-C are block diagrams illustrating the basic environment in which the motion control server system of the present invention to be used;

FIG. 1 is a module interaction map depicting the interaction of the primary modules of one exemplary server system of the present invention;

FIG. 2 is a scenario map illustrating the service discovery process implemented by the server system of FIG. 1;

FIG. 3 is a scenario map illustrating the machine configuration process implemented by the server system of FIG. 1;

FIG. 4 is a scenario map illustrating the machine monitoring process implemented by the server system of FIG. 1;

FIG. 5 is a scenario map illustrating the machine control process implemented by the server system of FIG. 1;

FIG. 6 is a module interaction map depicting the interaction of the primary modules of a data format module portion of the server system of FIG. 1;

FIG. 7 is an interface map illustrating the interface of the data format module of FIG. 6;

FIG. 8 is an object interaction map illustrating the interaction of the modules of the data format module of FIG. 6;

FIG. 9 is a scenario map illustrating the schema activation process implemented by the data format module of FIG. 6;

FIG. 10 is a scenario map illustrating the schema data query process implemented by the data format module of FIG. 6;

FIG. 11 is a scenario map illustrating the schema data set process implemented by the data format module of FIG. 6;

FIG. 12 is a scenario map illustrating the schema adding process implemented by the data format module of FIG. 6;

FIG. 13 is a scenario map depicting the basic transfer of a service request from a client application and the server system of FIG. 1;

FIG. 14 is a scenario map depicting the use of packet processing to transfer a service request response from the server system of FIG. 1 to a client application;

FIG. 15 is a scenario map depicting one exemplary initial connection process implemented by the server system of FIG. 1;

FIG. 16 is a scenario map depicting one exemplary method call process implemented by the server system of FIG. 1;

FIG. 17 is a scenario map depicting another initial connection process implemented by the server system of FIG. 1;

FIG. 18 is a scenario map depicting another exemplary method call process implemented by the server system of FIG. 1;

FIG. 19 is a module interaction map depicting the interaction of the primary modules of a service request format module of the server system of FIG. 1;

FIG. 20 is a module interaction map depicting the interaction of the primary modules of the service request format module of the server system of FIG. 1;

FIG. 21 is a scenario map depicting the initialization process implemented by the service request format module of FIG. 20;

FIG. 22 is a scenario map depicting the service request transfer process implemented by the service request format module of FIG. 20; and

FIG. 23 is a scenario map depicting the clean-up process implemented by the service request format module of FIG. 20.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention may be embodied as a motion control server system comprising a number of modules. In the following discussion, the overall environment in which the present invention is typically used will first be described. Following that will be a detailed discussion of the interaction of the various modules that form one exemplary embodiment of the present invention. The exemplary embodiment operates in a number of scenarios, and a number of these scenarios will then be described. Finally, certain components of the exemplary motion control server system, and typical use scenarios for these components, will be described in further detail.

I. OVERVIEW OF MOTION CONTROL SERVER SYSTEM

Referring initially to FIGS. 1A of the drawing, depicted at 20a therein is a motion control server system constructed in accordance with, and embodying, the principles of the present invention. The exemplary motion control server system 20a is configured to transmit motion control data between a data source 22 and a motion control system 24 through a communications system 26. The motion control data is represented by an application program 28 comprising methods, function calls, and/or data.

The exemplary motion control server system 20a comprises a service request format module 30 and a data format module 32. The service request format module 30 converts service requests (methods and/or function calls) of the application program 28 between a network service request format and a native service request format defined by the motion control system 24. The data format module 32 converts data sets transferred between the source 22 and the motion control system 24 between a network data format and a native data format defined by the motion control system 24.

FIGS. 1B and 1C indicate that some benefits of the present invention may be obtained by using either one of the service request format module 30 and the data format module 32. Depicted in FIG. 1B is an alternative exemplary motion control server system 20b that employs only the data format module 32, while FIG. 1C depicts yet another exemplary motion control server system employing only the service request format module 30.

II. EXEMPLARY MOTION CONTROL SERVER SYSTEM

Referring now to FIG. 1, depicted at 20 therein is one preferred embodiment of a motion control server system of the present invention. The motion control server system 20 will be described herein in the context of one exemplary data source 22, motion control system 24, and communications system 26. However, the present invention may be embodied in forms appropriate for other data sources, motion control systems, and communications systems. In addition, the preferred motion control server system 20 comprises a number of optional modules that are not necessary to carry out the principles of the present invention in a basic form.

Two sets of terminology will be used with reference to the motion control server system 20. The first set is generic and is applicable to any environment in which a motion control server system of the present invention may be used. The second is specific to the exemplary motion control server system 20 and the data source 22, motion control system 24, and communications system 26 in connection with which the server system 20 is used. In the following discussion, the major elements will be initially identified using the generic terminology, with the specific terminology being identified in parenthesis. After this initial introduction, both sets of terminology will be used interchangeably.

The exemplary motion control server system (XMC Internet Connect system) 20 comprises both the service request format module (XMC SOAP Engine) 30 and data format module (XMC XML Engine) 32. In addition, the exemplary server system 20 comprises two optional modules: a data caching module (XMC SQL Store) 40 and method discovery module (XMC DynaDiscovery) 42. These components allow virtually any client application 28 to utilize the underlying motion control system 24 regardless of the client application's location, underlying hardware platform, or underlying software operating system.

These modules 30, 32, 40, and 42 are optimized to connect the data source (client machine or device) 22 to a motion services module (XMC Motion Services) 50 forming a part of the motion control system 24 over the communications system (Internet) 26. The XMC Motion Services module 50 is a hardware independent connection to the underlying motion control hardware system (not shown). The exemplary XMC Motion Services module 50 is described in detail in one or more of the following U.S. Patents 5,691,897, 5,867,385, and 6,209,037 B1 and will not be described herein in further detail.

The exemplary XMC SOAP Engine module 30 is based on an industry standard technology referred to as SOAP (Simple Object Access Protocol). SOAP is an internet enabled communication protocol used to communicate in a platform and operating system independent manner with systems across the Internet. SOAP thus allows software applications to talk to one another in a manner that is independent of the communication connection or platform on which each application may be running. SOAP frees each application to run on the platform best suited for the application yet communicate with other systems as needed in a manner that connects all applications seamlessly. SOAP itself is based on two other industry standard technologies: HTML and XML. HTML defines an industry standard communication protocol for transferring data and instructions between applications connected to a network, while XML defines the

structure of the data packets sent between such applications. SOAP, HTML, and XML are well-known and will not be described herein in further detail.

The XMC XML Engine module 32 is used to translate network (XML) data sets into native (motion control) operations that are defined by and run on the XMC Motion Service 50 (also referred to as the native system). In addition, the XMC XML Engine 32 is used to query data from the native system 50 and build XML data sets that are returned to the calling client application 28.

The XMC SQL Store module 40 is used to cache data queried from the XMC XML Engine 32 (or directly from the native XMC Motion Services module 50). The exemplary XMC SQL Store module 40 caches data in database module 44 (SQL database or other database such as Microsoft Access or Oracle, etc).

The XMC DynaDiscovery module 42 is used to 'discover' the services supported by both the XMC XML Engine 32 and native XMC Motion Service module 50. The exemplary method discovery module 42 is based on the industry standard DISCO (Discovery of Web Services) protocol.

As noted in Figure 1 above, there are also several other modules that optional may be used with or incorporated into the XMC Internet Connection server system 20. In particular, the server system 20 uses the motion services (XMC Motion Services) module 50, motion drivers (XMC Driver) 52, a process control (XMC OPC) module 54, a packet processing (ROPE) module 56, and a data management (Biztalk Server system 2000) module 58.

The XMC Motion Services module 50 controls the motion control device to perform operations such as querying data, setting data, and performing actions to occur (like live physical moves). As generally discussed above, the XMC Motion Services module 50 is a hardware independent technology that supports many different hardware based and

software based motion controllers. The present invention may, however, be embodied without the motion services module 50 or its equivalent in a hardware dependent manner.

The motion services module 50 defines a group of supported motion control devices. One XMC Driver 52 is specifically written for each of the supported motion devices based on interfaces defined by the motion services module 50. The motion drivers 52 are known and will also not be described herein in detail.

The exemplary process control module 54 is a standard OPC (OLE for Process Control) server used to query and set data sets using the OPC protocols as defined by the OLE for Process Control Foundation.

The exemplary packet processing module 56 is a DLL module released by Microsoft and referred to as ROPE (Remote Object Proxy Engine). The ROPE module is specifically designed to build and parse SOAP data packets.

The exemplary data management module 58 is or may be the Microsoft BizTalk 2000 server. The Biztalk 2000 server is used to map data between XML Schemas, set up data agreements between companies, and manage data connections between organizations.

FIG. 1 further illustrates that the exemplary server system 20 employs a number of 'schemas' that are passed between modules. A 'schema' is a data format specification for XML data. Each schema determines how data following the protocol of the schema is organized. The exemplary server system 20 makes use of the following schemas: motion control (XMC) schemas 60, process control (OPC) schemas 62, database management (SQL) schemas 64, and third party schemas 66 such as the OMAC schema.

The XMC schemas are defined to support configuration data, system state data, motion meta program data, and actions defined by the XMC Motion Services module 50. The OPC Schema is an XML schema designed to support OPC servers. The SQL Schema is an XML schema

designed to describe SQL data sets queried from a SQL database. The OMAC Schema is designed to support data sets developed by the OMAC group.

One of the functions of the Microsoft BizTalk Server system 2000 module 58 is to map between schemas. Many groups and organizations will develop various data schemas that meet their particular needs. The Microsoft BizTalk Server system 2000 module 58 is capable of mapping between the schemas developed by different groups and organizations.

III. OPERATIONAL SCENARIOS

A. Service Discovery

Before a web service can be used, the services that service offers are determined or "discovered". Before discovering what a single web service can do, the web server is queried to determine what the web services that it offers. In the exemplary server system 20, the optional method discovery module 42 is used to discover the services available from the motion services module 50 using one or more of a plurality of protocols such as the Dynamic Web Discovery (DISCO) protocol, SNMP, LDAP, and the like. The exemplary XMC DynaDiscovery module 42 uses the DISCO protocol because the DISCO protocol is based on XML, which allows a very thin client to use the discovery service.

FIG. 2 of the drawing illustrates the steps that occur when the exemplary server system 20 uses the method discovery module 42 to discover the services available from the motion services module 50. First, the client application (or machine or device) 28, queries the motion control server system 20 for the services provided. This request may go through the BizTalk Server 58 or directly to the SOAP enabled server module 30.

If the request goes to the BizTalk Server 58, the BizTalk Server 58 maps the request to the appropriate format supported by the SOAP enabled server 30 and passes the request on to the SOAP server 30. The

BizTalk server 58 may also just pass the request straight through to the SOAP server if no mapping is needed.

Next, upon receiving the request, the XMC SOAP server 30 uses the ROPE module 56 to parse out the request. The XMC SOAP server module 30 could also use its own native parsing, but the use of the ROPE module 56 is preferred.

The XMC SOAP Server 30 next uses the XMC DynaDiscovery module 42 to determine what services are available on the local motion services module 50. Communication between the module 42 and the module 50 may be direct or may utilize an industry standard interface protocol such as a DCOM enabled connection; the interface protocol is schematically indicated by reference character 70 in FIG. 2.

Upon receiving the request, the XMC DynaDiscovery module 42 queries all modules that it 'knows about'. Such modules typically include or define type libraries (TLB) 72 that define the offered services. The exemplary module 42 thus examines the Type Libraries 72 to 'discover' the services that they offer. Upon discovering the available services, the DynaDiscovery module 42 dynamically builds an SDL (Services Description Language) data set and returns it to the requesting SOAP server 30. When dynamic discovery is not used, the SDL file is a static file that resides on the SOAP enabled server.

After determining what services are available from the motion services module 50, the client application program 28 may perform any of the operations offered by the module 50. When working with motion systems, these operations usually fall into one of three categories: configuration, monitoring/diagnostic, and actions. Each of these actions will be discussed in more detail below.

B. Machine Configuration

Configuration operations are used to configure the underlying motion system. For example, servo gains, velocities and accelerations may be set when performing configuration situations.

Configuration settings are usually separated into two categories: Initialization and Live Settings. Initialization configuration properties are usually set once when the machine first initialized. Live settings are changed dynamically to affect how the machine operates. The scenario discussed applies to changing both types of configuration settings.

Referring now to FIG. 3, depicted therein is a scenario map depicting the process of making configuration settings.

First, the client application 28 sends the configuration change request (along with all data needed to make the change) to the server system 20. This request may be sent to a BizTalk Server 58 or directly to the XMC SOAP Server 30, depending on the Schema used.

If the configuration change request is sent to the BizTalk Server 58, the BizTalk server 58 maps the data received from original schema to one of the schemas supported on the SOAP enabled server system 20; the Biztalk server 58 then sends the request on to the XMC SOAP Engine server 30. Upon receiving the SOAP request, the XMC SOAP Engine 30 optionally but preferably uses the ROPE module 56 to parse the request.

Next, the XMC SOAP Engine 30 passes the request data to the XMC XML Engine 32. The XMC XML Engine 30 configures the underlying native system (in this case the XMC Motion Service 50). The XMC SOAP Engine 30 may communicate with the XMC XML Engine 32 either locally or across a DCOM connection 70.

Depending on the schema used in the request, the XMC XML Engine 32 either uses the XMC OPC Server 54 to configure the native system 50 or configures the XMC Motion Services 50 directly. The XMC

XML Engine 32 may also use any other module to carry out the request as long as the XMC XML engine 32 has support for the external module's schema installed.

If the XMC OPC Server 54 is used, the XMC OPC server 54 then changes the configuration data as specified in the request made to it by the XMC XML Engine 32.

When requested, the XMC Motion Services 50 uses the current XMC Driver 52 to change the settings on the target motion hardware.

C. Machine Monitoring/Diagnostics

Monitoring/Diagnostic operations are used to query the system for information. For example when monitoring the system the current motor positions, velocities etc may be monitored to see what the machine is doing. When performing diagnostic operations, the actual state of the machine (such as the programs that reside on the hardware) may be queried. This information may be displayed to the user running the client, used for immediate analysis, or stored for future analysis. Machine diagnostics is similar to monitoring the machine except that a higher level of data detail is usually queried. The following scenario applied to both monitoring and querying diagnostic information from the machine.

Referring now to FIG. 4, the following steps occur when monitoring (or querying diagnostic information from) the machine.

First the client application 28 queries the machine (one time, intermittently, or periodically) for the information required. The client application 28 may use one of various different schemas. The data, configured according to the schema used, is sent to XMC SOAP Engine 30 either directly or indirectly through the BizTalk Server 58.

If the BizTalk Server 58 receives the request, it either directs the request to the XMC SQL Store module 40 or directly to the XMC SOAP Engine 30, depending on the schema mapping used and whether or not data caching is used.

If data caching is enabled, the XMC SQL Store module 40 queries the SQL database 44 (or any other database used) for the data. To update the cache, the XMC SQL Store module 40 either directly queries the XMC XML Engine 32 or uses the XMC SOAP Engine 30 to query the XMC XML Engine 32 for the data to be cached.

When requested, the XMC SOAP Engine 30 uses the ROPE engine 56 to parse the request and then either directly queries data specified in the request from the XMC Motion Services module 50, or routes the request to the XMC XML Engine 32.

If used, the XMC XML Engine 32 determines the data schema used and then either routes the request to the XMC Motion Services module 50 either directly or indirectly through the XMC OPC Server 54. If the XMC OPC Server 54 is used, it directly queries the data from the XMC Motion Services. The XMC Motion Services module 50 then uses the XMC Driver 52 to query the data from the target motion hardware.

D. Action Operations (Machine Control)

Action operations cause the machine to do things such as run programs or make moves. For example, the machine may be directed to move to its home state. The scenario depicted in FIG. 5 describes the process of performing such control operations.

The following steps occur when performing a machine control operation.

First, the client application 28 requests the machine control operation to be performed and passes all parameter data needed. This request is sent to the SOAP Enabled Server 30 directly or indirectly through the BizTalk Server 58.. The client application 28 may use one or more of various schemas to describe the operation to be performed.

If the BizTalk Server 58 is used, the BizTalk server 58 will, if necessary, map from the original schema used by the client application 28

2003-04-24 16:00

to a different schema supported by the motion server system 20. Once properly mapped, the request is passed to the XMC SOAP Engine 30.

When requested, the exemplary XMC SOAP Engine uses the ROPE module 56 to parse the request and determine what service operation is to be performed. As discussed above, the use of the ROPE module 56 is not required but is preferred.

Next, the XMC SOAP Engine 30 sends the request to the XMC Motion Services module 50 for processing either directly or indirectly through the XMC XML Engine 32. As discussed above, this communication may be on a local machine or may occur across a DCOM connection 70 (or even to another SOAP Engine if appropriate). If used, the XMC XML Engine 32 connects with the XMC Motion Services module 50 to perform the requested machine control operations.

The XMC Motion Services module 50 uses the selected XMC Driver 52 to direct the target hardware to perform the desired machine control operations.

IV. DATA FORMAT MODULE

The data format, or XMC XML Engine, module 32 acts as a container for multiple schemas, both XMC XML and non-XMC XML Schemas. The XMC XML Engine thus forms a schema repository that is available to other components of the system 20 such as the service request format module 30 or the data management module 58.

Once enabled with several schemas, the XMC XML Engine 32 uses polymorphism to work with each schema in the same manner. Each schema itself is the definition of a data set used to either query or set motion configuration data and motion properties, or cause motion actions on the target machine.

This section describes the how the XMC XML Engine works internally as well as how it interacts with the modules around it in a software system.

A. XML Engine Module Interactions

The XMC XML Engine 32 is designed to be a 'middle-ware' component that translates data between a native system and XML. In the exemplary system 20, this translation is bi-directional. Translations from the XML data format to the data format of the native motion services module 50 are used to change configuration data and properties and to cause actions. Translations from the native data format of the motion services module 50 to XML data format are used when querying configuration data or properties.

FIG. 6 is a module interaction map illustrating the interaction of the XMC SOAP Engine 30 and the XMC XML Engine 32 when the SOAP Engine 30 calls methods on the XML Engine 32. The methods called allow the XMC SOAP Engine 30 to query and set data or cause other actions on the native system implemented by the XMC Motion Services module 50.

As noted above, the XMC XML Engine 32 may work with several native systems. FIG. 6 illustrates that the XMC XML Engine 32 also can work with the XMC OPC component 54 to query/set data sets using the OLE for Process Control data formats.

Even though only the XMC Soap Engine 30 is displayed as the only client, many other clients could use the XMC XML Engine 32. For example, a Microsoft BizTalk server 58 might be used to query specific data from the XMC XML Engine 32 and then map that data into a completely different schema, such as the OMAC data schema 66.

FIG. 6 illustrates that the exemplary XMC XML Engine module 32 interacts with the XMC SOAP Engine 30, the XMC Motion Services module 50, and the XMC OPC module 54. As generally discussed above, the XMC XML Engine module 32 is used to build data sets based on the active XML Schema 60. In addition, this engine 54 translates data sets

received and enables requested operations, such as setting configuration or property data, to be performed by the native motion control system 24.

Referring now to FIG. 7, depicted at 80 therein is an interface map for the XMC XML Engine module 32. The exemplary XMC XML Engine module 32 implemented as a COM component that houses several objects. Each object exposes one or more OLE interfaces.

FIG. 7 illustrates that the XMC XML Engine module 32 houses two primary objects: the SchemaContainer object 82 and SchemaEnum objects 84. In addition, the XMC XML Engine 32 supports several default Schema objects 86, although an infinite number of external Schema objects can be supported. When querying and setting schema data, the SchemaContainer object 82 is used because it aggregates to the active Schema object. The SchemaEnum object 84 is used to enumerate across all Schema objects installed.

The SchemaContainer object 82 manages all Schema objects installed and is the main object used by the client applications 28. The container 82 stores all Schema objects 86, one of which is designated as the active schema. The SchemaContainer object 82 contains the IXMCSchemaContainer interface, the IXMCPersistSchemaContainer interface, and the IXMCSchema interface. The IXMCSchemaContainer interface is used to add/remove schema objects and get access to the schema enumeration. In addition, this interface allows the caller to activate one schema or another. The IXMCPersistSchemaContainer interface is used to persist all information with regard to the schema collection, including the active schema. The IXMCSchema interface is an aggregation of the active schema object's IXMCSchema interface.

The SchemaEnum object is responsible for enumerating across the set of installed schema objects and contains the IXMCEnumSchema interface. The IXMCEnumSchema interface is a standard COM enumeration interface.

5 The Schema objects are responsible for implementing the specific schema supported. To support a schema, the schema object is responsible for translating Native System data into the XML data set defined by the schema. In addition, XML data sets may be translated and used to change configuration and property settings in the native system.
10 And finally, XML data sets may be translated and used to cause actions on the native system such as physical moves.

The Schema objects define the IXMCSchema interface and IPersist interface. The IXMCSchema interface allows clients to Set and Query data sets based on the XML schema supported.

15 The XMC XML Engine object interactions will now be described with reference to FIG. 8. In particular, FIG. 8 illustrates how the COM components making up the XMC XML Engine interact to service client requests with data supported by several different schemas.

20 As shown in FIG. 8, the Schema Container object 82 is the main object that manages all other objects. Client applications may interact with each object directly at times, but the Schema Container object 82 is one of the first that the client application will encounter.

25 The Schema Container object 82 gives the client application access to the Schema Enumerator object 84 used to enumerate across all schema objects 86 installed. Access to the Schema Enumerator object 84 is useful when working with several different schemas at the same time or when browsing the set of installed schemas. For example, if the Schema Container object 82 has schemas installed that support OPC, XMC and OMAC objects or data sets 86, the enumerator object 84 allows the calling
30 application to enumerate across each of these objects 86.

From the Schema Container object 82, the client application may also install new Schemas 86 and set the active schema out of those installed. Specifying the one of the schema 86 as the active schema directs the Schema Container 82 to aggregate the IXMCSchema interface from the specified active schema object so that the active schema 86 may be used in future data query/set/action operations.

Specifying, selecting, or "activating" a schema is the process of directing the Schema Container to make a schema in a group of previously installed schema the active schema. The 'active' state means that the Schema Container 82 aggregates the specified schema's IXMCSchema interface so that all calls to the Schema Container 82 appears to be housing this object; in actuality, the Schema Container routes the interface to the actual schema object.

The process of "activating" a schema will now be described in further detail with reference to FIG. 9. Initially, the calling client application 28 using the XMC XML Engine 32 calls the Schema Container object 82 and directs the object 82 to specify one of the support schema as the active schema. A special ID, GUID, text name, or some other unique identifier may identify each schema. This identifier is used to tell the Schema Container which schema to activate.

Once notified, the Schema Container 82 uses its internal Schema Enumerator 84 to query for the specified schema. If the specified schema is not found an error is returned.

Upon finding the target schema, the Schema Container 82 aggregates the IXMCSchema interface of the activated Schema object 86, making it appear to the client application 28 that the Schema Container 82 actually implements the activated Schema object 86.

Once a schema 86 is activated, the client application 28 may choose to query data from the active schema. Such a query may be used to query all configuration settings on a machine or query the overall state

of the machine. FIG. 10 illustrates the steps that take place when querying data.

First, the client application 28 queries the Schema Container 82 for the data from the active Schema object. Upon receiving the request, the request actually routes directly to the active Schema object 86 through the aggregated IXMCSchema interface.

Based on the schema supported, the Schema object 86 queries the native system (in this case the XMC Motion Server 50) for all data needed to fill out the data request. The data required to fill out the data request is then packaged into an XML data packet as specified by the supported Schema. The XML data packet is then passed back to the calling client application 28.

In addition to querying data, the native system configuration and properties may also be set or actions may be performed. Setting data on the native system is very similar to the reverse of the querying process.

In particular, FIG. 11 illustrates the steps that occur when setting data on the native system.

First, the client application sends a 'set' request to the Schema Container 82, making sure to pass the XML data packet specifying the data to be set. Upon receiving the request, the call is routed directly to the active Schema object 86 through the aggregated connection to the active Schema's IXMCSchema interface. The Schema object then parses the XML data packet based on the Schema that it supports.

As data is parsed from the XML data packet (or upon completing the parsing), the Schema object 86 directs the native system (in this case the XMC Motion Server 50) to set all data items specified. If an action is requested, the Schema object 86 would parse the data packet pulling from it the data parameters to pass along to the native system 50 when directing it to perform the action requested. The action requested would be specified as part of the data packet. For example, an action identifier

may be used to specify an operation to perform from a set of supported operations.

Upon completing the request, the system 20 returns the status (success or failure) of the operation to the client application 28.

To use schemas other than the set of default schemas supported by the XMC XML Engine 32, the client application must add new ones.

FIG. 12 illustrates the steps that occur when adding new schema support to the Schema Container. Initially, the client application must request the Schema Container 82 to add a new schema 86, making sure to specify the CLSID (or other identifier) of the schema and URL (or other location identifier) identifying the location of the new Schema object 86.

Upon receiving the request, the Schema Container 82 creates an instance of the new Schema object 86 and adds it to its list of supported schemas. When persisting its information, the Schema Container 82 saves the schema identifier and location so that it can later load the schema object.

B. Schema Examples

This section shows several example schemas, each of which would be supported by one or more Schema objects 86.

1. XMC Configuration Schema

The XMC configuration schema describes all data used to configure an XMC software system.

```
<?xml version='1.0' encoding='UTF-8' ?>
```

```
<!ELEMENT XMCCConfiguration (Systems)>
```

```
<!ATTLIST XMCCConfiguration Version CDATA #IMPLIED >
```

```
<!ELEMENT Systems (System+)>
```

<!ATTLIST Systems Count CDATA #IMPLIED >

<!ELEMENT System (DefUnits , DefMode , SecurityOptions , Drivers)>

<!ATTLIST System Number CDATA #IMPLIED >

5 <!ELEMENT DefUnits (#PCDATA)>

<!ELEMENT DefMode (#PCDATA)>

<!ELEMENT SecurityOptions (Security.control , Security.monitoronly)>

<!ELEMENT Security.control (#PCDATA)>

10 <!ELEMENT Security.monitoronly (#PCDATA)>

<!ELEMENT Drivers (Driver+)>

<!ATTLIST Drivers Count CDATA #IMPLIED >

15 <!ELEMENT Driver (Enabled , Filters , Properties , Streams)>

<!ELEMENT Filters (Filter*)>

<!ATTLIST Filters Count CDATA #IMPLIED >

20 <!ELEMENT Filter (Streams)>

<!ELEMENT Properties (Property*)>

<!ATTLIST Properties Count CDATA #IMPLIED >

25 <!ELEMENT Property (Name , Value)>

<!ELEMENT Streams (Stream*)>

<!ATTLIST Streams Count CDATA #IMPLIED >

30 <!ELEMENT Stream (Enabled , (Stream.PCBus | Stream.TextFile |
Stream.DbgMon))>

<!ELEMENT Stream.PCBus (Port , PortSize)>

<!ELEMENT Stream.TextFile (FileName)>

<!ELEMENT Stream.DbgMon EMPTY>

<!ELEMENT FileName (#PCDATA)>

<!ELEMENT Name (#PCDATA)>

5 <!ELEMENT Value (#PCDATA)>

<!ELEMENT Enabled (#PCDATA)>

<!ELEMENT Port (#PCDATA)>

<!ELEMENT PortSize (#PCDATA)>

10 2. XMC Meta Programs Schema

The XMC Meta Program schema describes data making up a meta program which is a hardware independent motion control program.

15 <?xml version='1.0' encoding='UTF-8' ?>

<!ELEMENT XMCMetaProject (Programs)>

<!ATTLIST XMCMetaProject Version CDATA #IMPLIED >

<!ELEMENT Programs (Program*)>

20 <!ATTLIST Programs Count CDATA #IMPLIED >

<!ELEMENT Program (Name , Commands)>

<!ATTLIST Program SystemNum CDATA #IMPLIED >

25 <!ELEMENT Commands (Command*)>

<!ATTLIST Commands Count CDATA #IMPLIED >

<!ELEMENT Command (Name , Parameters)>

30 <!ELEMENT Parameters (Parameter*)>

<!ATTLIST Parameters Count CDATA #IMPLIED >

<!ELEMENT Parameter (#PCDATA)>

<!ATTLIST Parameter Type CDATA #IMPLIED >

<!ELEMENT Name (#PCDATA)>

3. XMC System State Schema

5

The XMC System State schema is used to query/set all aspects of the motion control system.

<?xml version='1.0' encoding='UTF-8' ?>

10

<!ELEMENT XMCState (Configuration, Axes, Programs, RawData, ErrorStatus)>

<!ATTLIST XMCState Version CDATA #IMPLIED >

15

<!ELEMENT Configuration (Config.ActiveCFGFile , Config.ActiveMPFFFile)>

<!ELEMENT Config.ActiveCFGFile (#PCDATA)>

<!ELEMENT Config.ActiveMPFFFile (#PCDATA)>

20

<!ELEMENT Axes (Axis+)>

<!ATTLIST Axes Count CDATA #IMPLIED >

<!ELEMENT Axis (CommandedData, ActualData, HomingData, Limits , State)>

<!ATTLIST Axis Index CDATA #IMPLIED >

25

<!ELEMENT CommandedData (Commanded.MotionProfile, Commanded.Position)>

30

<!ELEMENT ActualData (Actual.MotionProfile, Actual.MotorPosition, Actual.EncoderPosition)>

<!ELEMENT Commanded.MotionProfile (MotionProfile)>

<!ELEMENT Homing.MotionProfile (MotionProfile)>

<!ELEMENT Actual.MotionProfile (MotionProfile)>

```

5      <!--ELEMENT HomingData (Homing.MotionProfile ,
        Homing.FinalVelocity)>
      <!--ELEMENT Homing.FinalVelocity (#PCDATA)>

      <!--ELEMENT MotionProfile (Velocity , Acceleration , Deceleration)>
      <!--ELEMENT Velocity (#PCDATA)>
      <!--ELEMENT Acceleration (#PCDATA)>
      <!--ELEMENT Deceleration (#PCDATA)>

10     <!--ELEMENT Commanded.Position (#PCDATA)>
      <!--ELEMENT Actual.Position (#PCDATA)>
      <!--ELEMENT Actual.MotorPosition (#PCDATA)>
      <!--ELEMENT Actual.EncoderPosition (#PCDATA)>

15     <!--ELEMENT RawData (RawData.Programs , RawData.Configuration)>
      <!--ELEMENT RawData.Programs (#PCDATA)>
      <!--ATTLIST RawData.Programs DataSize CDATA #IMPLIED >
      <!--ELEMENT RawData.Configuration (#PCDATA)>

20     <!--ATTLIST RawData.Configuration DataSize CDATA #IMPLIED >
      <!--ELEMENT Limits (Limits.IsHit_HWCCW,
        Limits.IsHit_HWCW,
        Limits.IsHit_SWCCW,
        Limits.IsHit_Home,
        Limits.IsHit_SWCW,
        Limits.SWCCWPos,
        Limits.SWCWPos )>

25     <!--ELEMENT State (IsMoving , IsHoming , IsFaulted)>
      <!--ELEMENT IsMoving (#PCDATA)>
      <!--ELEMENT IsHoming (#PCDATA)>
      <!--ELEMENT IsFaulted (#PCDATA)>

30     <!--ELEMENT ErrorStatus (Error.Internal , Error.Source)>

```

<!ELEMENT Error.Internal (Error)>

<!ELEMENT Error.Source (Error)>

<!ELEMENT Error (#PCDATA)>

<!ATTLIST Error ErrorCode CDATA #IMPLIED >

<!ELEMENT Programs (Program+)>

<!ATTLIST Programs Count CDATA #IMPLIED

ActiveProgram CDATA #IMPLIED >

<!ELEMENT ActiveProgram (#PCDATA)>

<!ELEMENT Program (Name)>

<!ATTLIST Program IsRunning CDATA #IMPLIED

Index CDATA #IMPLIED >

<!ELEMENT Name (#PCDATA)>

4. XMC OPC Schemas

In addition to the XMC specific schemas previously described, non XMC schemas may also be supported. This section shows OLE for Process Control schemas designed for motion. The XMC OPC schema is actually an OPC schema designed for XMC data that is formatted specifically for an OPC server.

<?xml version='1.0' encoding='UTF-8' ?>

<!ELEMENT Server (Groups)>

<!ATTLIST Server Version CDATA #IMPLIED >

<!ELEMENT Groups (Group+)>

<!ATTLIST Groups Count CDATA #IMPLIED >

<!ELEMENT Group (Type , UpdateRate , Items)>

<!ELEMENT Type (#PCDATA)>

<!ELEMENT UpdateRate (#PCDATA)>

5

<!ELEMENT Items (Count , Item+)>

<!ATTLIST Items Count CDATA #IMPLIED >

<!ELEMENT Item (ID , Description , DataType , Data)>

10

<!ELEMENT ID (#PCDATA)>

<!ELEMENT Description (#PCDATA)>

<!ELEMENT DataType (#PCDATA)>

<!ELEMENT Data (#PCDATA)>

15

<!ELEMENT Count (#PCDATA)>

V. SERVICE REQUEST FORMAT MODULE

20

This section contains further description of the SOAP (Simple Object Access Protocol), how SOAP is implemented in the context of the data server system 20, and how to setup the service request format module 30 in the context of the XMC SOAP Engine.

25

To request an operation in another application, the client application sends an HTML 'POST' instruction containing an XML 'SOAP Envelope'.

The XML Soap Envelope defines the operation to be performed.

30

Referring initially to FIG. 13, depicted therein is the basic HTML Soap request as implemented using the data server system 20 of the present invention. To operate over a communications network 24 such as the Internet, the data server system 20 must be capable of receiving Internet/Web requests. FIG. 1 illustrates this capability by an internet information application programming interface (IIAPI) 74. FIG. 13 illustrates that the interface 74 is defined by an information server module 76; in the exemplary system 20, the information server module 76 is

formed by a Microsoft Internet Information Server (IIS) based server installed with the XMC SOAP Engine 30.

Upon receiving a request, the information server module 76 passes the request to the XMC Soap Engine 30, which in turn performs the requested motion operation. Once complete, the Server responds with a HTML header and XML 'SOAP Envelope' that describes the results of the operation.

In addition to using basic HTML, a data packet processing technology is available from Microsoft Corporation called 'Remote Object Proxy Engine' or ROPE. ROPE performs all basic tasks of generating the HTML/XML SOAP data packets sent to the server system 20 when requesting operations. In addition ROPE parses the responses retrieved from the server system 20. While optional, the use of the ROPE technology is preferred.

FIG. 14 illustrates the process of using ROPE technology for parsing of packets sent to and retrieved from the server system 20. ROPE builds and sends the same HTML 'POST' instruction with the same SOAP Envelope containing the XML data describing the requested operations and any parameters used.

When making a SOAP request, a particular sequence of steps must be performed to carry out the request. This sequence of steps forms what will be referred to herein as the "SOAP Pipeline". The SOAP Pipeline will be described below from two different perspectives. In the first, the pipeline is described making a SOAP request just using basic HTML on the client side. The second scenario describes making a SOAP request using the Microsoft ROPE technology.

A. SOAP Request Using Basic HTML

SOAP requests are available to any client application that supports HTML and XML. This section describes the specific steps taking place when making a basic HTML based SOAP request.

Initial connection is not required, but is helpful in that establishing an initial connection informs the client machine about the services available on the SOAP Server. If the client is informed in advance about the services that are available, the initial connection step is not necessary.

Referring now to FIG. 15, the HTML Connect process will be described in further detail.

When making the initial connection, following steps take place.

First, the client must build a standard HTML 'GET' header used to query the 'services.xml' file that contains the 'Services Description'. This is often referred to as the SDL or Services Description Language and is an XML based document that describes the operations available on the SOAP enabled server. The file containing the Service Description Language document may be given any name but must be an XML file.

Next, the client must send the HTML request to the server to query the server for the XML services file. Upon receiving the request, the server returns the services description (the contents of the services.xml file) to the client. The client may then parse the services description to 'learn' what services are supported by the SOAP server (including the method and parameter names and types).

Once the client 22 has identified the services available on the SOAP server 30, the client 22 is ready to make method calls directing the server to perform the supported operations.

FIG. 16 illustrates the process of making an HTML Method Call. The client must first build a standard HTML 'POST' header specifying the host and 'SoapAction', where the SoapAction includes both the location of the '*.SOD' file and the service requested. The SOD file describes the actual COM component that will be used to carry out the operation, whereas the service requested is the method exposed by that component.

Next, the client application 28 must build the SOAP envelope that describes the service requested. Using XML, the envelope is built to describe the method and all parameter data used to perform the service request.

5 The client then sends the request to the SOAP enabled server system 20, making sure to send the request to the location where the XMC SOAP Engine 30 is installed. Upon receiving the request, information server module 76 routes the .SOD based request to the XMC SOAP Engine 30 for processing.

10 On the Server side, the XMC SOAP Engine 30 uses the ROPE module 56 to load and parse the .SOD file to get the component object to use for the request. In addition, the ROPE module 56 is used to parse the actual XML contained within the request that describes the SOAP operation to be performed (ie method name and parameters).

15 The XMC SOAP Engine 30 then actually makes the call to the component method passing all parameters sent via the previous SOAP call.

20 Next, the XMC SOAP Engine 30 again uses the ROPE module 56 to build the response packet, making sure to build into the packet the results of the component method call. If any data is to be returned (as in the case of querying the component, such as with the XMC XML Engine 32), the data is packed into the response SOAP envelope using the ROPE module 56.

25 The ROPE module then sends the response SOAP envelope back to the client application 28. Upon receiving the response, the client application 28 may parse the HTML and XML to get the response results and queried data, if any.

30 The previous sections have described communicating to a SOAP enabled motion control server system 20 using native HTML. One skilled in the art will recognize that, in both the general GET and POST operations, the client was required to build the header and SOAP request

Envelope as well as parse the SOAP response envelope. The next section describes how using the ROPE module 56 simplifies significantly these steps because the ROPE module 56 handles all parsing and envelope building programmer.

5 The Remote Object Proxy Engine (ROPE) was designed specifically to build and parse SOAP envelopes. In addition, ROPE takes care of sending each packet to the server as well as receiving responses from the server. This section describes the same SOAP pipeline but from the perspective of using ROPE instead of native HTML.

10 When using ROPE, the initial connection between the client application 28 and the server system 20 is required, for this connection identified for the ROPE module 56 what services are available on the SOAP enabled server system 20.

15 FIG. 17 illustrates the steps that occur when making the initial connection using ROPE.

20 First, using the SOAPPackager object, the client application 22 loads the service description by passing the services description file ('services.xml') URI to the LoadServiceDescription method. Internally, the SOAPPackager object builds the "get" request and sends this request to the SOAP enabled server system 20. This is the same get request described in the native HTML initial connection.

 Upon receiving the request, the information server module 76 responds by sending back the contents of the services.XML file.

25 The SOAPPackager object is then used on the client side to parse out the listener information, which is the URI of the services.SOD file on the server module 76. At this point, the SOAPPackager object has all the information necessary to determine what services are available on the server, along with the specific format of each service.

30 Once the initial connection is made, the client application 22 is able to use the ROPE 56 to invoke services (make method or service request calls) on the SOAP enabled server system 20.

As shown in FIG. 18, the following steps occur when invoking services using the ROPE module 56.

Using the SOAPPackager object, the local copy of the service description is loaded (this is the same one previously loaded but cached by ROPE). The SOAPPackager object is then used to build the payload for the service that is to be called by specifying the 'method name' of the service. In addition, the SOAPPackager object is used to add the parameter data for the method call, if any such parameter data is required.

Using the WireTransfer object, the standard SOAP headers are added to build the actual HTML header and SOAP envelope that will eventually be sent. This is the same HTML 'POST' header and SOAP envelope described above with calling methods using native HTML.

The WireTransfer object is then used to send the header and SOAP envelope containing the service request to the server system 20, thereby requesting the that the server system 20 instruct the motion control system 24 to perform the contained service request.

Upon receiving the request, information server module 76 detects the .SOD based request and routes the request to the XMC SOAP Engine 30. The XMC SOAP Engine 30 uses the local ROPE module 56 to parse the COM component name from the .SOD file as well as parse out the service request information from the XML SOAP envelope contained in the original request.

Next, the XMC SOAP Engine 30 calls the method on the XMC Motion Server 50 as requested by the service request contained in the original SOAP envelope.

All results from the service request (method) invocation and all return data are packed into a response SOAP envelope built by the ROPE module 56. The response SOAP envelope is the returned to the client application 22 by the ROPE module at the XMC SOAP Engine 30. The client application 22 then uses the SOAPPackager object to parse the

response SOAP envelope and make available to the client application 22 all return parameters contained in the response SOAP envelope.

A close comparison of the steps discussed with reference to FIGS. 16 and 17 indicates that the use of the ROPE module 56 eliminates many of the native HTML based SOAP steps by generating and parsing the SOAP envelope for the programmer.

With the foregoing understanding of how the client application 28 interacts with the SOAP enabled server system 20, the construction and operation of the server system 20 will now be described in detail.

The XMC SOAP Engine 30 builds on SOAP technology to obtain the data server system 20 that is enabled for motion-based application. In particular, the exemplary XMC SOAP Engine 30 extends information server module 76 to support SOAP requests and routes each request appropriately to the method on the component implementing the service requested. The following sections describe how the XMC SOAP Engine 30 performs these tasks to support SOAP requests.

The XMC SOAP Engine 30 handles SOAP requests received through the Internet 26. Such requests may originate from any client application 22 or may actually be routed to the XMC SOAP Engine 30 from a BizTalk server 58.

In particular, the XMC SOAP Engine 30 interacts with several modules in the system 20 as will be described below with reference to FIG. 19.

Similar to any other SOAP client, the Microsoft BizTalk server 58 may send SOAP requests to the XMC SOAP Engine 30 as well to request data that as necessary to fill out data within supported schemas. For example, a BizTalk server 58 may be used to map an OMAC schema 62 to an XMC schema 60. When filling out the data in the OMAC schema 62, the BizTalk server 58 may query data from the XMC SOAP Engine 30 to fill out the end data mapping.

In addition, other clients may talk to the XMC SOAP Engine 30 via the Internet as previously discussed in the sections above describing the SOAP Pipeline.

To fulfill SOAP requests, the XMC SOAP Engine 30 works with both the XMC XML Engine 32 and with the XMC Motion Server 50. Data queries and configuration settings are made using the XMC XML Engine 32, and service requests are carried out directly by the XMC Motion Server 50 or indirectly through the XMC XML Engine 32.

The exemplary XMC SOAP Engine 30 comprises several objects. These objects work together to perform each requested SOAP operation. In addition, the XMC SOAP Engine 30 uses the XMC Motion Server 50 to eventually carry out the actual service request, either directly or using the ROPE module 56.

The exemplary XMC SOAP Engine 30 is a standard extension module for the Microsoft Internet Information module 74. As such, the XMC SOAP Engine 30 exposes the GetExtensionVersion, HttpExtensionProc, and TerminateExtension functions. These functions are called by module 74 on each request.

Referring now to FIG. 20 of the drawing, that figure shows that the XMC SOAP Engine 30 comprises a CsoapApp object 90, a GetExtensionVersion module 92, an HTTPExtension module 94, a TerminateExtension module 96, a Thread Pool 98 comprising one or more worker threads 98a, and a CsoapRequest module 100.

The CSoapApp object 90 manages each of the extension DLL entry points and routes each request appropriately to either the thread pool 98 or the CSoapRequest object 100. In addition, the CsoapApp object 90 is responsible for creating and destroying the worker thread pool 98.

The CSoapRequest object 100 is responsible for managing the data describing the actual service request. A new object is created for each service request and passed to a worker thread 98a for processing.

The thread pool 98 is a collection of threads 98a each of which is used to process one service request.

As generally described above, the ROPE DLL module 56 is used to parse each SOAP envelope and also to build the response SOAP envelopes that are sent back to the client application 28.

As generally discussed above, the XMC Motion Server 50 and XMC XML Engine 32 are used to carry out the requested operations (ie data query, configuration setting, or motion action).

Before the XMC SOAP Engine 30 can be used, it must be initialized. Initialization occurs on the first request when information server module 76 first loads the extension DLL.

The following steps occur during the XMC SOAP Engine Initialization. Upon receiving the first service request, the information server module 76 loads the extension DLL and calls the GetExtensionVersion module or function 92. Upon receiving this call, the CSoapApp object 90 creates the thread pool 98.

When processing a service request, the XMC SOAP Engine 30 creates a CSoapRequest object 100 and passes it to one of the threads 98a in the thread-pool 98 for processing. The thread 98a then in turn directs the specific motion operations to occur on the XMC Motion Server 50.

Referring now to FIG. 22, the following steps occur when the XMC SOAP Engine 30 processes a SOAP request.

First, upon receiving the SOAP service request, the information server module 76 calls the HttpExtensionProc 94, passing along all information about the service request. Inside the function call, the CSoapApp object 90 is used to process the request.

When called, the CSoapApp object 90 creates a CSoapRequest object 100 and passes to it the service request information. Next, the CSoapApp object 90 passes the new CSoapRequest object 100 to a free thread 98a in the thread pool 98 and directs the thread 98a to start

processing the request. To process the request, the worker thread 98a first accesses the CSoapRequest object 100 passed thereto by the CsoapApp object 90.

Next, the worker thread 98a uses the ROPE module 56 to parse the response and get the PROGID of the designated component to be used to carry out the request.

The designated object or component specified by the request is accessed from either the XMC Motion Server 50 or the XMC XML Engine 32, and the appropriate method is called based on the SOAP request.

Once the method completes, the result and any data returned is packed into a SOAP response envelope and sent back to the client application 28.

Upon termination of the motion control server system 20, the information server module 76 shuts down the XMC SOAP Engine 30. During this process, the XMC SOAP Engine 30 frees all resources used. This clean-up process will now be described in further detail with reference to FIG. 23.

Initially, upon termination of the motion control server system 20, the information server module 76 terminates the extension DLL by calling its TerminateExtension function 96. When called, the CSoapApp object 90 destroys the worker thread pool 98.

The following discussion will describe how to setup the XMC Soap Engine 30 to run with Microsoft Internet Information Server 76 on a Windows 2000 system. The requirements for the setup process are a Windows NT 2000 Server with NTFS formatted hard drives and Microsoft Internet Information (IIS), version 5.0 or above. Internet Explorer 5.0 or above is recommended but is not required.

This section describes how to configure IIS for use with the XMC Soap Engine. To setup IIS, a virtual directory is created where the XMC SOAP engine 30 is to run. When creating the virtual directory, the following settings should be followed:

Application Protection	Low (IIS Service)	Run the programs in the virtual directory (ie the XMC SOAP Engine and all COM components that it uses) with the IWAM_<machname> user account access level.
Read Access	Enable	Turn on read access so that data files (ie the service.xml and service.sod files) can be read.
Execute Permissions	Scripts & Executables	Allow scripts and executables to run (ie the XMC Soap Engine and all COM objects that it uses).
Directory Security	Defaults (Anonymous, Integrated Windows authentication)	Use the default directory security settings.

Next, the XMC Soap Engine IIS Extension is installed and several NT services are prepared to run with the XMC Soap Engine 30. The following sections describe each of these tasks. Please note that the virtual directory must be placed on an NTFS file system and the services.xml and services.sod files must be granted both Read and Execute access.

To setup the XMC Soap Engine ISAPI Extension, the 'Configuration...' button is selected from the 'Properties' page for the virtual directory. On the 'App Mappings' tab, select the 'Add' button to add a new mapping.

Browse for the location of the XMCSOAP.DLL and enter the location into the 'Executable' field. Make sure the location of the XMC Soap Engine 30 is the full path of the file on a Local hard drive; the access level at which the engine 30 runs does not have network access. Next, enter '*.sod' as the 'Extension'. Both the 'All Verbs' and 'Script engine' check boxes should be selected.

This mapping associates the *.sod file extension to the XMC Soap Engine ISAPI Extension DLL. Once mapped, the XMC Soap Engine

ISAPI Extension DLL is called by IIS each time IIS encounters a file with the extension .sod from within the virtual directory.

Both the IIS Admin Service and World Wide Web NT services must have 'interact with use' access. To enable each service with this access level, open the 'Computer Management' console by selecting the 'Start | Programs | Administrative Tools | Computer Management' menu item. From the console, select the 'Services and Applications | Services' tree item.

Next for both the 'IIS Admin Service' and 'World Wide Web' services, the following steps are performed. First, the service is opened by double clicking. The 'Log on' tab is then selected. The 'Local system account' radio button is next selected. Finally, the 'Allow service to interact with the desktop' check box is selected, just below the 'Local system account' radio button

Since the XMC Soap Engine uses several COM components and NT services. Each of these services should be configured in the following manner to allow proper interaction with the XMC Soap Engine 30.

Using DCOMCNFG.EXE the COM security level on all components as well as on the specific components used by the XMC Soap Engine shall be configured by making the following default properties using DCOMCNFG.EXE:

Setting	Value	Description
Enable Distributed COM on this computer	Yes	This will allow the NT services to talk COM objects.
Enable COM Internet Services on this computer	No	Not used.
Default authentication level	Connect	
Default impersonation level	Identity	

In addition, the following default security settings should be made using DCOMCNFG.EXE:

5

Setting	Value	Description
Default Access Permissions	None set	For extra security, these will only be set on the specific servers.
Default Launch Permissions	IUSR_<machinename>	Internet Web User (browsing the site)
Default Launch Permissions (cont.)	IWAM_<machinename>	IIS Process access level.

Each XMC Motion executable must be configured using the DCOMCNFG.EXE utility as well. In particular, the following XMC binaries must be configured: XMCSRVC.EXE and XMCDBGWN.EXE.

All other XMC modules (which are DLLs) will run under the IIS Process security access level.

For each of the executables listed above, the following security settings should be made using the DCOMCNFG.EXE utility.

Setting	Value	Description
Default Access Permissions	IWAM_<machinename>	Internet Web User (browsing the site).
Default Launch Permissions	IUSR_<machinename>	Internet Web User (browsing the site)

10

As a final security check, each and every EXE and DLL used by the XMC Soap Engine (including the XMC Soap Engine) must have both Read and Execute file permissions. All server files MUST be installed on a local hard drive for they will be accessed from within the IIS Process, which does not have network access.

15

Similar to the IIS Admin Service and World Wide Web service, the XMC Service must be configured to *'Allow service to interact with the desktop'*.